Computer Programming 2
(Structures)

Dr. Dina M. Ibrahim
1st year Students
2014-2015

Lecture #4

07/3/2015

Structures

1. Combining Data into Structures
2. Accessing Structure Members
3. Initializing a Structure
4. Nested Structures
5. Structures as Function Arguments
6. Returning a Structure from a Function
7. Examples

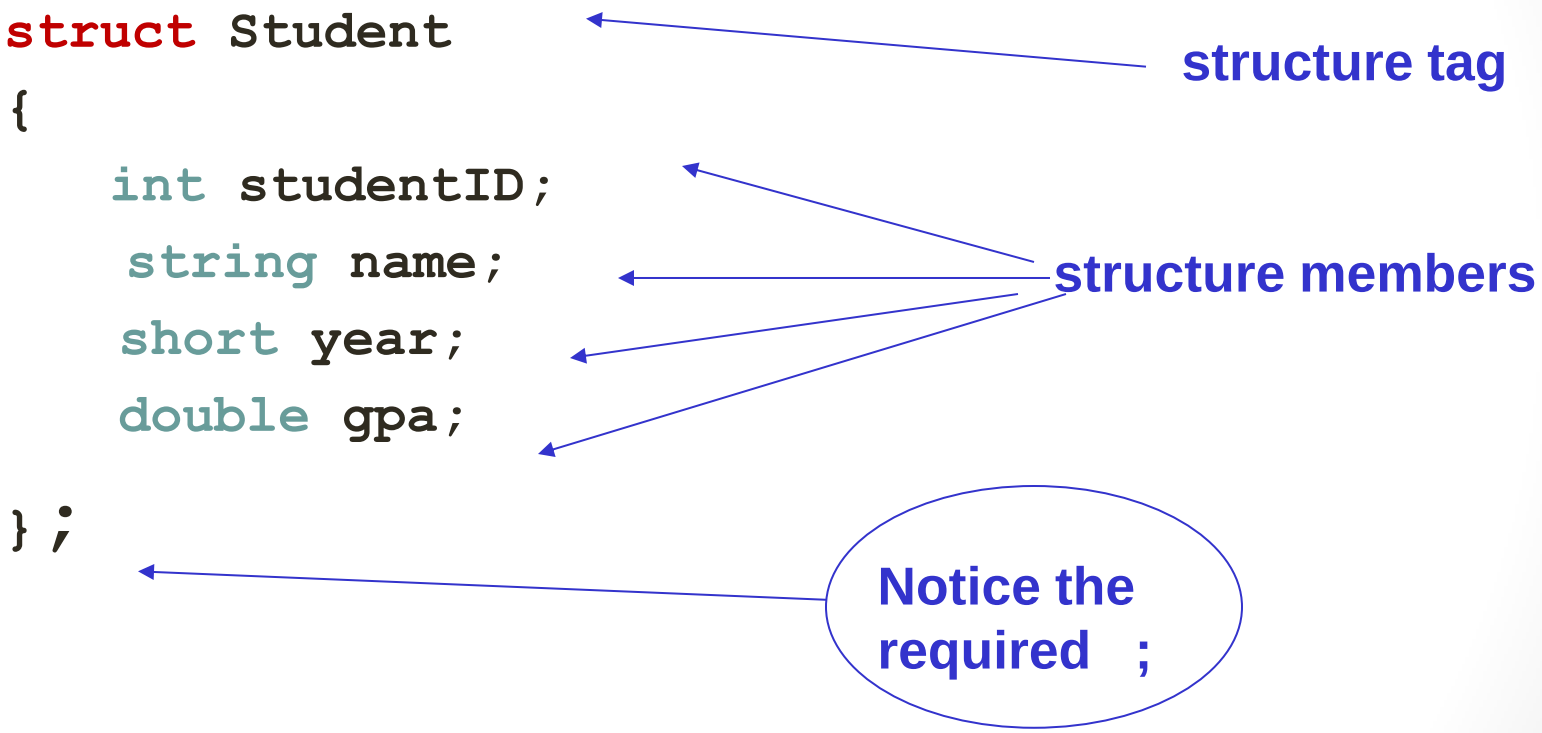
1. Combining Data into Structures

- Structure: is a C++ construct that allows multiple variables to be grouped together

- Structure Declaration Format:

```
struct structure_name
{
  type1 field1;
  type2 field2;
    …
  typen fieldn;
};
```

Example `struct` Declaration



`struct` Declaration Notes

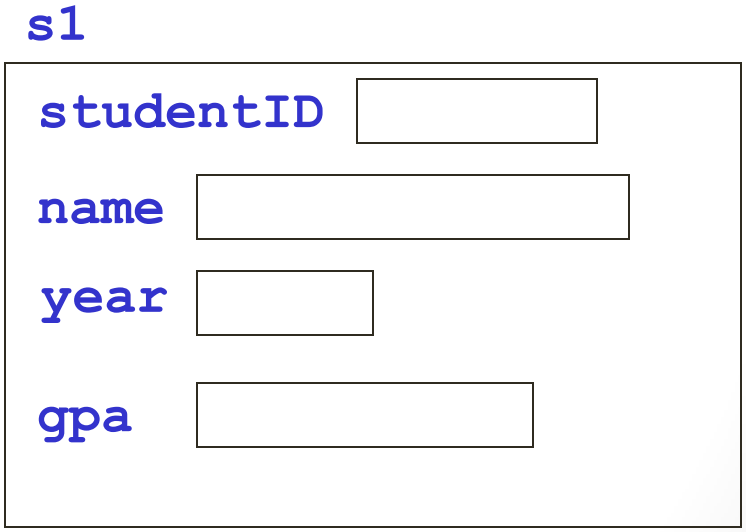
- **`struct`** names commonly begin with an uppercase letter
- The structure name is also called the tag
- Multiple fields of same type can be in a comma-separated list:

```
string name, address;
```

Defining Structure Variables

- **`struct`** declaration does not allocate memory or create variables
- To define variables, use structure tag as type name

```
Student s1;
OR
   struct Student
   {
      int studentID;
      string name;
       short year;
       double gpa;
   }s1;
```



2. Accessing Structure Members

- Use the dot **(.)** operator to refer to members of **struct** variables

```
cout<<"Enter the student name: ";
getline(cin, s1.name);
cout<<"Enter the student ID: ";
cin >> s1.studentID;
s1.gpa = 3.75;
```

- Member variables can be used in any manner appropriate for their data type

Displaying `struct` Members

- To display the contents of a **`struct`** variable, you must display each field separately, using the dot operator

Wrong:

```
cout << s1; // won't work!
```

Correct:

```
cout << s1.studentID << endl;
cout << s1.name << endl;
cout << s1.year << endl;
cout << s1.gpa;
```

Comparing `struct` Members

- Similar to displaying a **`struct`**, you cannot compare two **`struct`** variables directly:

```
if (s1 >= s2) // won't work!
```

- Instead, compare member variables:

```
if (s1.gpa >= s2.gpa) // better
```

3. Initializing a Structure

- Cannot initialize members in the structure declaration, because no memory has been allocated yet

```
struct Student // Illegal initialization
{
   int studentID = 1145;
   string name = "Ali";
   short year = 1;
   float gpa = 2.95;
};
```

Initializing a Structure (continued)

- Structure members are initialized at the time a structure variable is created.
- Can initialize a structure variable's members with either
  - an initialization list
  - a constructor

Using an Initialization List

- An initialization list is an *ordered* set of values, separated by commas and contained in **{ }**, that provides initial values for a set of data members

```
{101, "Ahmed", 3, 3.8}   // initialization list
                         // with 4 values
```

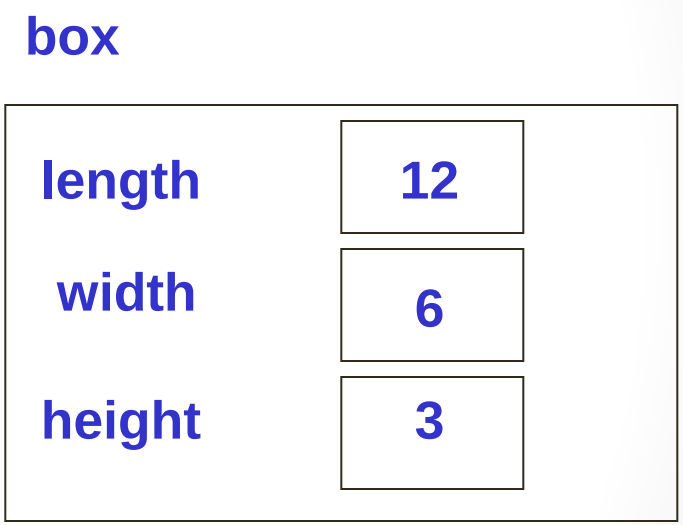
Initialization List Example

Structure Declaration

```
struct Dimensions
{ int length,
      width,
      height;
};

Dimensions box = {12,6,3};
```

Structure Variable



Partial Initialization

Can initialize just some members, but cannot skip over members

```
Dimensions box1={12,6};  //OK
Dimensions box2={12,,3}; //illegal
```

Using a Constructor to Initialize Structure Members

- A constructor is a special function that can be a member of a structure
- It is normally written inside the **`struct`** declaration
- Its purpose is to initialize the structure's data members

Using a Constructor (continued)

- Unlike most functions, a constructor is not *called;* instead, it is automatically invoked when a structure variable is created
- The constructor name must be the same as the structure name (i.e. the struct tag)
- The constructor must have no return type

A Structure with a Constructor

```
struct Dimensions
{
   int length,
       width,
       height;
   // Constructor
   Dimensions(int L, int W, int H)
   {length = L; width = W; height = H;}
};
```

Passing Arguments to a Constructor

- Create a structure variable and follow its name with an argument list.
- Example:

```
Dimensions box3(12, 6, 3);
```

Default Arguments

A constructor may be written to have **default arguments**

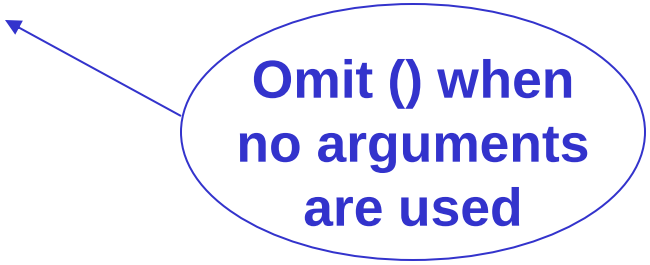
```
struct Dimensions
{
   int length,
       width,
       height;
   // Constructor with default values
   Dimensions(int L=1, int W=1, int H=1)
   {length = L; width = W; height = H;}
};
```

Structure Constructor Examples

```
//Create a box with all dimensions given
Dimensions box4(12, 6, 3);

//Create a box using default value 1 for
//height
Dimensions box5(12, 6);

//Create a box using all default values
Dimensions box6;
```



4. Nested Structures

- A structure can have another structure as a member.

```
struct PersonInfo
{  string name,
           address,
           city;
};

struct Student
{  int          studentID;
   PersonInfo   pData;
   short        year;
   double       gpa;
};
```

Members of Nested Structures

Use the dot operator multiple times to access fields of nested structures

```
Student s5;
s5.pData.name = "Ahmed";
s5.pData.city = "Tanta";
```

5. Structures as Function Arguments

- May pass members of **`struct`** variables to functions

```
computeGPA(s1.gpa);
```

- May pass entire **`struct`** variables to functions

```
showData(s5);
```

- Can use reference parameter if function needs to modify contents of structure variable

6. Returning a Structure from a Function

- Function can return a **`struct`**

```
Student getStuData();   // prototype
s1 = getStuData();      // call
```

- Function must define a local structure variable
  - for internal use
  - to use with **`return`** statement

Returning a Structure Example

```
Student getStuData() // Function Definition
{
   Student s;     // local variable
   cin >> s.studentID;
   getline(cin, s.pData.name);
   getline(cin, s.pData.address);
   getline(cin, s.pData.city);
   cin >> s.year;
   cin >> s.gpa;
   return s;
}
```

Example 1 (No. 1 in Sheet 3):

Create a structure data type called **Rectangle**. The structure has two members length and width. Provide the following functions:

1. Area( ) to calculate the area of a rectangle.
2. Perimeter( ) to calculate the perimeter of a rectangle.
3. Printing rectangle members in the form (a,b) where a is the length and b is the width.

Write a C++ program to test your structure and functions?

Answer of Example 1:

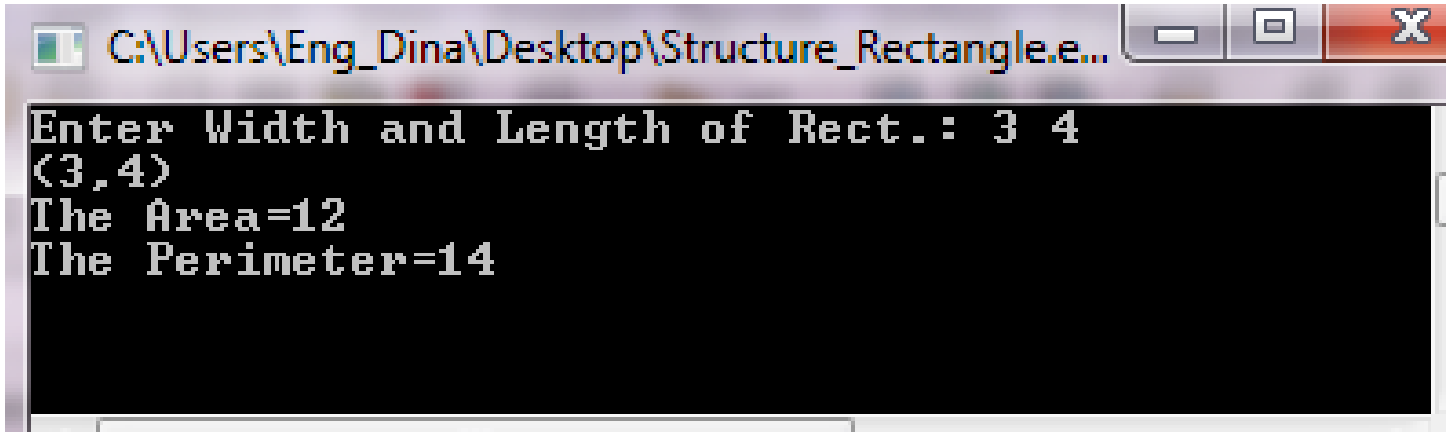
```
// Structure example
#include <iostream>
using namespace std;

struct Rectangle{
float width;
float length;
};

float Area_Rectangle(float w, float l)
{
return w*l;
}
float per_Rectangle(float w, float l)
{
return 2*(w+l);
}
void print(float w, float l){
cout<<"("<<w<<","<<l<<")"<<endl;
}
```

```
int main( )
{
Rectangle r;
cout<<“Enter Width and Length of Rect.: ";
cin>>r.width>>r.length;
float area, perimeter;
area=Area_Rectangle(r.width, r.length);
perimeter=per_Rectangle(r.width, r.length);
print(r.width, r.length);
cout<<“The Area="<<area<<endl;
cout<<“The Perimeter="<<perimeter;
}
```

Output of Example 1:



Example 2 (No. 2 in Sheet 3):

Create a structure data type called **Complex** for performing arithmetic with complex numbers. Provide the following functions:

1. Addition of two complex numbers.
2. Subtraction of two complex numbers.
3. Printing complex members in the form (a,b) where a is the real part and b is the imaginary part.

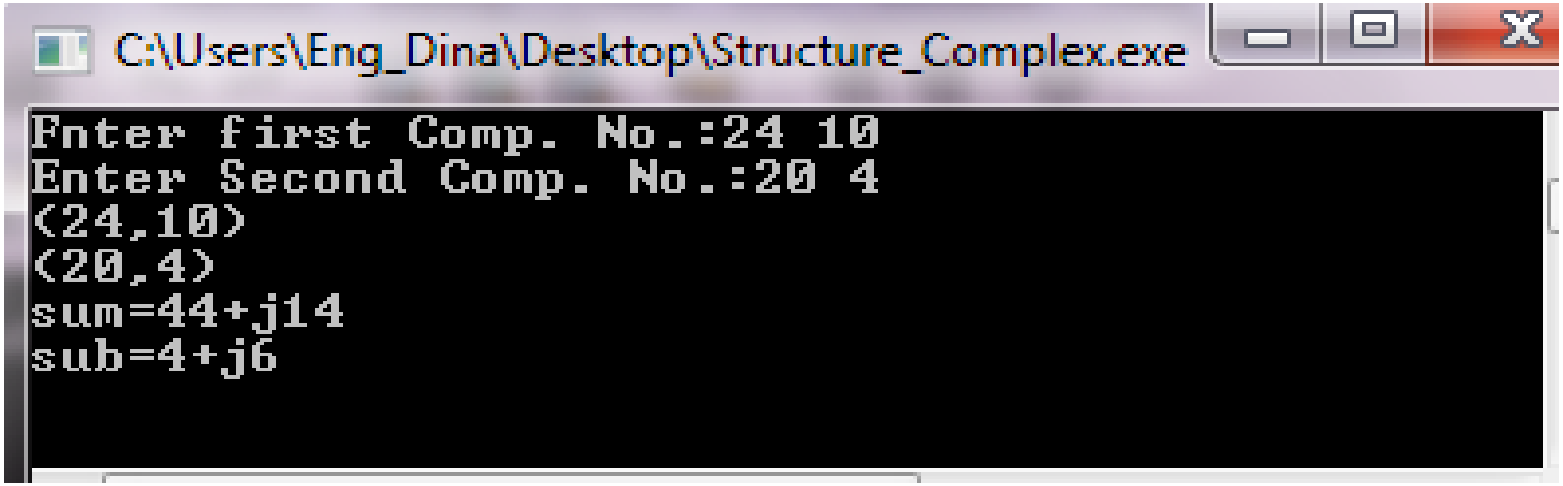
Write a C++ program to test your structure and functions?

Answer of Example 2:

```
#include <iostream>
using namespace std;
struct Complex{
float real;
float img;
};
Complex addition(Complex c1,Complex c2)
{
  Complex c;
  c.real=c1.real+c2.real;
  c.img=c1.img+c2.img;
  return c;
}
Complex subtraction(Complex c1,Complex c2)
{
  Complex c;
  c.real=c1.real-c2.real;
  c.img=c1.img-c2.img;
  return c;
}
void print_Complex(Complex c)
{ cout<<"("<<c.real<<","<<c.img<<")"<<endl;
}
```

```
int main( ){
Complex c1,c2;
Complex add, sub;
cout<<“Fnter first Comp. No.:";
cin>>c1.real>>c1.img;
cout<<“Enter Second Comp. No.:";
cin>>c2.real>>c2.img;
add=addition(c1,c2);
sub=subtraction(c1,c2);
print_Complex(c1);
print_Complex(c2);
cout<<"sum="<<add.real<<"+j”
     <<add.img<<endl;
cout<<"sub="<<sub.real<<"+j“
     <<sub.img;
return 0;
}
```

Output of Example 2:



Thanks